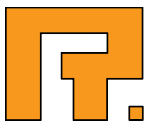

Roxen WebServer 5.1

If Tags Tutorial

2010-12-17
www.roxen.com

Roxen Internet Software
Box 449
SE-581 05 Linköping
Sweden
www.roxen.com

Contents

1 Introduction

Welcome to the Roxen CMS Tutorials. This section is dedicated to all users of Roxen CMS. The tutorials are intended for both beginners and experienced users, and we hope that all find some interesting reading and get creative ideas.

It is assumed that the reader is familiar with HTML and have some knowledge of XML.

As always, if you have any suggestions, comments or complaints regarding these tutorials do not hesitate to send an email to manuals@roxen.com and if the issue is an obvious bug do not hesitate to report it to Bug Crunch, our bug tracking system.

2 If Tags Tutorial

Welcome to the RXML If tags tutorial!

As you surely know, a typical web page consists of text and HTML tags sent over the Internet from a web server to a browser. The HTML tags tell the browser how the page should be displayed.

The Roxen Macro Language, RXML, offers a number of tags which are used in the same way as HTML tags, but extend the sometimes quite limited power of HTML. One group of RXML tags are the If tags. These tags make it possible to create dynamic pages based on conditions. You could let authenticated users only get confidential information of a page or optimize pages for different kinds of browsers. If tags also make it possible to create web applications in RXML without using any programming language.

Hopefully this brief presentation has made you curious about the powers of If tags. If so, please don't hesitate to read the following Sections of this Lesson.

This Lesson is designed so that you may move around as you please. Feel free to read only the Sections that interest you.

After reading this Lesson you will be able to program web pages using If tags and you will know some of their many useful features. As in most creations, a tutorial isn't enough to become a master. Only hard work will get you there...

2.1 The basics of if-else statements

This section will introduce the logic of if-else statements in general to beginners in programming. If you have some experience in programming you can skip this section.

After reading this section you will have knowledge of the basics of if-else statements.

We will create a simple web page with two radiobuttons and a regular button that will let you choose to display the text "Hello World!" in different styles. The example is rather meaningless in real life but is good to introduce you to the basics of if-else statements.

2.1.1 The basics

When programming you often want to control the order in which the statements will be executed. This is done by using Control Flow Statements, and some of those are the if-else statements.

The if-else statements enable your program to selectively execute other statements, based on some criteria. The simplest version, the if statement, is shown below. The block governed by if (delimited with '{' and '}') is executed if the expression is true, otherwise the execution continues after the last '}'.

```
if (expressions)
{
  statement(s)
}
```

If you want to execute other statements when the expression is false, you use the else statement.

```
if (expression)
{
  statement(s) executed if expression is true
}
else
{
  statement(s) executed if expression is false
}
```

Another statement, elseif, executes statements if an earlier if expression was false and it's own expression is true; elseif is used to form chains of conditions. If expression 1 is true, the if block executes and then the program jumps to the last '}' of the else block. Expression 2 will never be evaluated. If, however, expression 1 is false, expression 2 will be evaluated and the elseif-else will work as a regular if-else as shown above.

```
if (expr 1)
{
  statement(s) executed if expr 1 is true
}
elseif (expr 2)
{
  statement(s) executed if expr 1 is false and expr 2 is true
}
else
{
  statement(s) executed if expr 1 is false and expr 2 is false
}
```

2.1.2 A basic example

Let's do something real to examplify what have been mentioned so far. We will create a simple HTML page containing RXML (Roxen Macro Language) that will be rather meaningless except for demonstrating the basics of if and else. The file will show the text "Hello World!" either in plain text or bold text, depending on which radiobutton is checked by the user. Here is the code followed by screenshots of the output in a browser:

```
<html>
<head>
<style type='text/css'>
<!--
    body{background-color:#FEFEC9}
    h1{background-color:#FEED87}
  -->
</style>
</head>

<body>

<!-- HTML FORM -->

<h1>A basic example</h1>

<p>Check radiobutton and click &quot;OK&quot; for bold.</p>
```

```
<form action='hello_world.html' method='GET'>
<input type='radio' name='style' value='plain' />
 Plain style <br />
<input type='radio' name='style' value='bold' />
 Bold style <br />
<input type='submit' value='OK' />
</form>

<p>------------------------</p>

<!-- RXML CODE -->

<if variable='form.style is bold'>
<p><b>Hello World!</b></p>
</if>
<else>
<p>Hello World!</p>
</else>

</body>
</html>
```

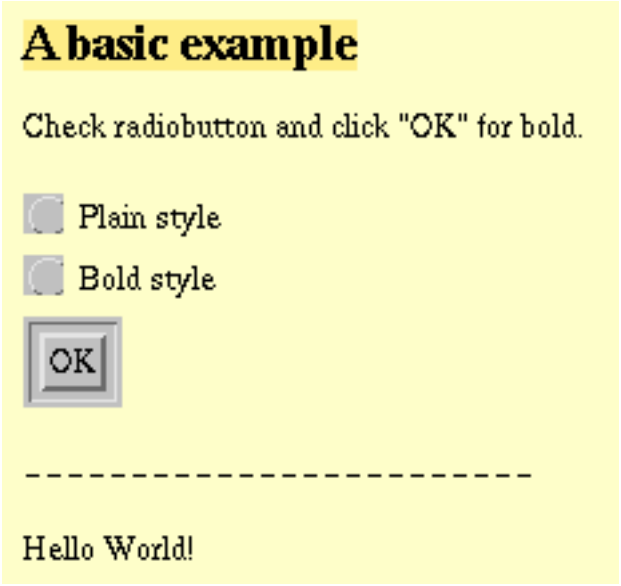

Figure 1. The page in the browser before any interaction.

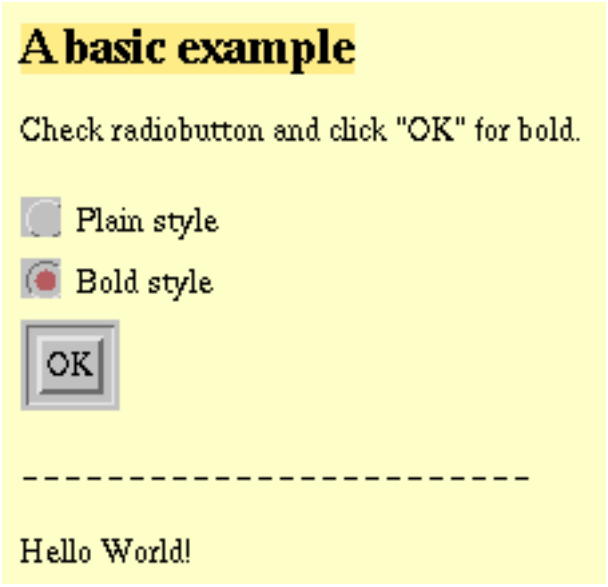

Figure 2. The user chooses 'Bold style', clicks OK...

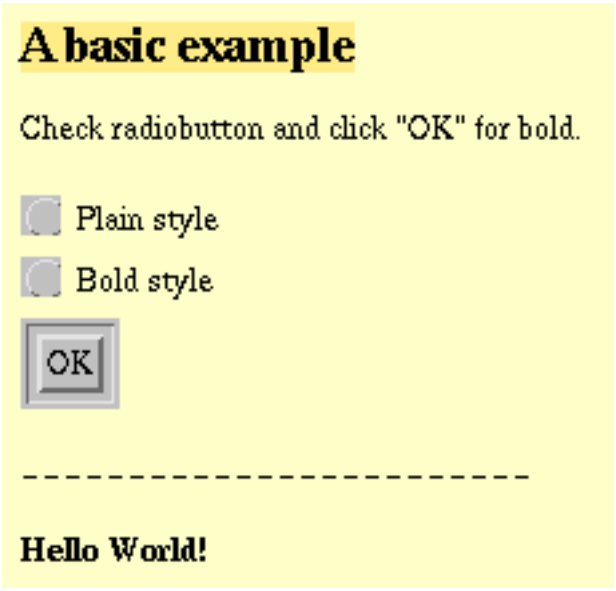

Figure 3. ...and the text goes bold.

The interesting part of the code is the `<!-- RXML CODE -->` section. We want to check if the user chose bold or not. When the user clicks the OK button, the variable style will contain either the string 'bold' or the string 'plain' (or perhaps be empty, if something goes wrong). We use an if-else statement to check which. If style contains 'bold', the expression `variable='form.style is bold'` will be true, the line inside if executes and 'Hello World!' will be bold. If style doesn't contain 'bold', the expression will be false and the line inside else will execute; 'Hello World!' will be plain text.

```
<!-- RXML CODE -->

<if variable='form.style is bold'>
<p><b>Hello World!</b></p>
</if>
<else>
<p>Hello World!</p>
</else>
```

Let us add the possibility to make the text italic. We insert the lines:

```
<input type='radio' name='style' value='italic' />
 Italic style <br />
```

below the second <input> and rewrites the RXML part to

```
<if variable='form.style is bold'>
<p><b>Hello World!</b></p>
</if>
<elseif variable='form.style is italic'>
<p><i>Hello World!</i></p>
</elseif>
<else>
<p>Hello World!</p>
</else>
```

This gives us the following result:

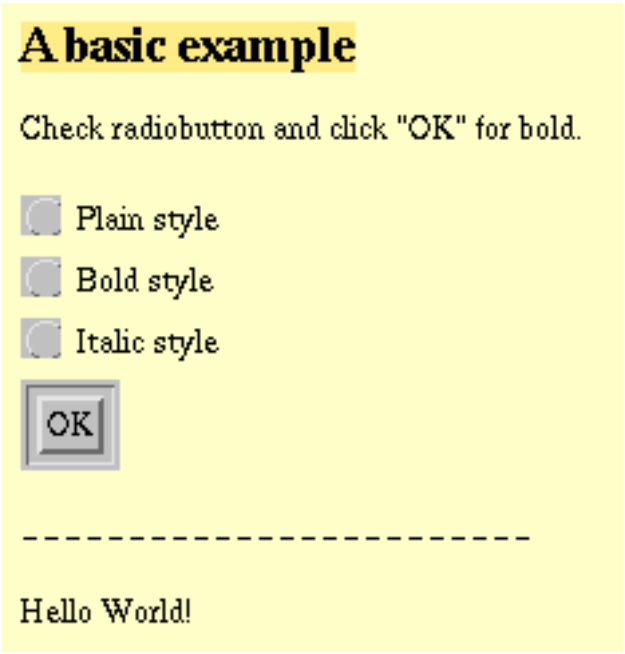

Figure 4. Now the user has three options..

As you might have guessed, this is an example of the if-elseif-else statement. If style contains 'bold', the if tag executes, if it contains 'italic', elseif executes and in all other cases, else executes.

2.1.3 Summary

This section has taught you the basics of the if-else statements in general. If-else statements is used to control the flow of a program. The if statement test a condition and if the condition is true, the if statement block will execute. The else statement will execute if an if condition is not true. The elseif statement is used to form chains of conditions.

More details about if-else statements and other control flow statements are found in any book or on any site that is teaching programming. However, the information in this Section should be enough to take you through the rest of this Lesson. The next section, The syntax of If tags will introduce the RXML If tags, including tags, attributes and operators.

2.2 The syntax of If tags

This section will introduce the basics of the RXML If tags.

After reading this section you will know the proper usage of RXML If tags using attributes and operators.

If you are looking for an example, see A basic example of <if>.

2.2.1 Tags

If-statements in RXML are built up by combining the six basic tags <if>, <else>, <elseif>, <then>, <true> and <false>. With <if> and <elseif>, attributes are used to state which test(s) to do. One attribute should be one of several If plugins. Logical attributes and, or and not are used to further specify the test(s). We sum this up in the following general syntax (code within '[ ]' is optional):

```
<if plugin1='expr' [and|or plugin2='expr' ...] [not]>
  if block
</if>
[<elseif plugin='expr' ...>
  elseif block
</elseif>]
[<else>
  else block
</else>]
```

or

```
<if plugin1='expr' [and|or plugin2='expr' ...] [not] />
<then>
  if block
</then>
[...]
```

Always remember to close tags

```
<if variable='var.foo = 10' />
```

or

```
<if variable='var.foo = 10'></if>
```

and to add values to attributes

```
<if true=''></if>
----------------------
<if variable='var.foo = 1' and='' match='&var.bar; is bar'>
```

for proper XML syntax. This is necessary since RXML is a XML compliant language.

Let's have a look at the basic tags:

<if>

is used to conditionally show its contents; provides a framework for the If plugins.

<else>

shows its contents if the previous <if> returned false, or if there was a <false> above.

<elseif>

is the same as <if>, but will only evaluate if the previous <if> returned false.

<then>

shows its contents if the previous <if> returned true, or if there was a <true> above.

<true>

is an internal tag used to set the return value of If tags to true.

<false>

is an internal tag used to set the return value of If tags to false.

```
<set variable='var.foo' value='1' />

<if variable='var.foo = 1'>
  var.foo is 1
</if>
<else>
  var.foo is not 1
</else>
```

Results in:

```
var.foo is 1
```

A test is made if the variable var.foo is 1. This is true because the first line does nothing less than sets that variable to 1. Please note that the spaces around the '=' operator are mandatory.

```
<set variable='var.foo' value='1' />

<if variable='var.foo = 1' />
<then>
  var.foo is 1
</then>
<else>
  var.foo is not 1
</else>
```

Results in:

```
var.foo is 1
```

The same test using if-then-else instead.

```
<true />
<then>
  truth value is true
</then>
<else>
  truth value is false
</else>
```

Results in:

```
truth value is true
```

In this example the internal `<true>` is used to set the truth value to true so that the following `<then>` will be executed.

2.2.2 Attributes

The attributes used with `<if>` are:

plugin name

The If plugin that should be used to test something, e.g. <if variable>. It is mandatory to specify a plugin. See the If plugins Section for further information.

not

Inverts the result (true->false, false->true).

or

If any criterion is met the result is true.

and

If all criterions are met the result is true. and is default.

```
<set variable='var.foo' value='1' />

<if variable='var.foo = 1' not=''>
   var.foo is not 1
</if>
<else>
   var.foo is 1
</else>
```

Results in:

```
var.foo is 1
```

Here the test is logically negated with not.

The use of `<if variable='var.foo != 1'>` gives the the same result as `<if variable='var.foo = 1' not=''>`.

A common mistake done is when combining and, or and not.

```
<if user='foo' or='' not='' domain='*.foobar.com'>
  ...
</if>
```

will not work since the not attribute negates the whole tag, not just the domain attribute. If you want to negate the whole condition, add not at the end. If you only want to negate one of the attributes, you must rewrite the code with an if-elseif-else statement.

```
<if user='foo'>
</if>
<elseif domain='*.foobar.com' not=''>
  ...
</elseif>
```

```
<set variable='var.length' value='3' />
<set variable='var.string' value='foo' />

<if variable='var.length > 0' <b>and=''</b> match='&var.string; = foo'>
   var.string is 'foo'
</if>
<else>
   Either string is empty, doesn't contain string 'foo' or both.
</else>
```

Results in:

```
var.string is "foo"
```

A multiple test with two different If plugins, *variable* and *match*.

You could be tempted to write expressions like:

```
<if variable='var.foo = 1' or='' variable='var.bar = 1'>
  ...
</if>
```

This will not work, as you can only use the same attribute once. Here Variable is used twice.

2.2.3 Operators

Above we used the '>' operator. The operators that may be used in expressions are:

Operator	Meaning
=	Equals
==	Equals
Is	Equals
!=	Not equals
<	Less than
>	Greater than

Note that <= and >= are not possible operators except in the *Expr* plugin. However, for the effect of `<if variable='var.foo <= 1'>` you simply use `<if variable='var.foo > 1' not=''>` instead.

Global patterns are possible to use in almost all plugin expressions. ' * ' match zero or more arbitrary characters while ' ? ' matches exactly one arbitrary character.

```
<if ip='130.236.*'>
  Your domain is liu.se
</if>
<else>
  Your domain isn't liu.se
</else>
```

Results in:

```
Your domain isn't liu
```

In this example 130.236.1 as well as 130.236.123 would be true. If the test would be `<if ip='130.236.?'>` only 130.236.0 - 9 would be true.

Regular expressions are not supported in If tags.

2.2.4 Summary

This section has taught you the basics of the If tags. If statements are built up by the six basic tags, `<if>`, `<else>`, `<elseif>`, `<then>`, `<true>` and `<false>`.

With `<if>` and `<elseif>` an attribute naming an *If* plugin must be added. The general syntax is:

```
<if plugin1='expr' [and|or plugin2='expr' ...] [not]>
  if block
</if>
[<elseif plugin='expr' ...>
  elseif block
</elseif>]
[<else>
  else block
</else>]
```

Logical attributes, *and*, *or* and *not* adds functionality. Inside the attribute expression, the '=', '==', 'is', '!=', '<' and '>' operators are valid.

Always remember to close tags (/) and to give attributes a value (=") for proper XML syntax.

More details about *If* tags is found in the Web Site Creator Manual or by putting `<help for='if' />` in a web page.

The next section, If plugins will explain the usage of the *If* plugins.

2.3 If plugins

This section will introduce the If plugins used as attributes with `<if>` and `<elseif>`.

After reading this section you will have knowledge of the different plugin categories and the usage of the plugins.

The following sections will contain examples of how to use different plugins for creating dynamic web pages in many different ways.

2.3.1 The categories

The If plugins are sorted into fourfive categories according to their function: *Eval*, *Match*, *State* and *Utils*, and *SiteBuilder*.

Eval	Match	State	Utils	Sitebuilder
clientvar	accept	config	date	ppoint
Cookie	client	false	exists	
Expr	domain	pragma	group	
Match	ip	prestate	time	
Variable	language	supports	user	
	referrer	true		

The following parts will go through these categories and the corresponding plugins. We will look at the proper way of usage and some traps you might fall into.

2.3.2 Eval

The Eval category is the one corresponding to the regular tests made in programming languages, and perhaps the most used. They evaluate expressions containing variables, enities, strings etc and

are a sort of multi-use plugins. All If-tag operators and global patterns are allowed (see The syntax of If tags).

Plugin	Syntax
clientvar	clientvar='var[is value]'
Cookie	cookie='name[is value]'
Expr	expr='pattern'
Match	match='pattern1[,pattern2,...]'
Variable	variable='name[is pattern]'

Cookie and *Variable* plugins use a similar syntax. `is` can be any valid operator and `name` is the name of any defined or undefined variable(cookie). They first check if the named variable(cookie) exists, and if it does and a pattern(value) is specified, the expression will be evaluated. *Variable* is a general plugin while *Cookie* is for testing existence and/or values of cookies.

```
<set variable='var.foo' value='10' />

<if variable='var.foo = 10'>
  true
</if>
<else>
  false
</else>
```

Results in:

```
true
```

Please note that it is the name of the variable, not the entity representing it, that should be used. Here we receive an error from the RXML parser becase we used an entity instead of a variable name. The proper way to do the test above is `<if variable='var.foo = 10'>`.

Match is used in a similar way, but there are certain differences. The syntax is `<if match="pattern">`, where pattern could be any expression. We could use `<if match='var.foo = 10'>` although this is rather meaningless, since we would check if the name of the variable `var.foo` (or the string "var.foo") was equal to the string "10", which it obviously is not. The only thing that would return true in this case is another meaningless expression: `<if match="var.foo is var.foo">`.

Instead we should always use entities with *Match*, i.e. `&var.foo;`. In RXML, entities are used to represent the contents of something, e.g. a variable. An entity is written as a variable name enclosed with '&' and ';'. Above the name of the variable is var.foo, the value is '10' and the entity is `&var.foo;`. The entity is replaced by its content when parsed to HTML, in this case '10'.

```
<set variable='var.foo' value='10' />

if match='var.foo = 10'>
  true
</if>
<else>
  false
</else>
```

Results in:

```
true
```

Again, variable name goes with Variable and entity goes with *Match*.

A warning when testing patterns with whitespaces. If you want to test if the entity `&var.foo;` equals the string 'he is nice' and do like this...

```
<set variable='var.foo' value='he is nice' />
'&var.foo;' <br /><br />
<if match='&var.foo; is he is nice'>
  var.foo is 'he is nice'
</if>
<else>
  var.foo is not 'he is nice'
</else>
```

Results in:

```
"he is nice" var.foo is not "he is nice"
```

...you won't get the expected result. This is because the Match plugin interprets the first valid operator after a whitespace as the operator to use. In the example above the test really is if 'he' equals 'nice is he is nice', which obviously is false. Rember that the string 'is' also is a valid operator.

Expr evaluates mathematical and logical expressions. The following characters only should be used in the expression:

1, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **0**, **.**, **x**, **a**, **b**, **c**, **d**, **e**, **f**, **X**, **A**, **B**, **C**, **D**, **E**, **F**

For numbers, decimal, octal and hexadecimal. E.g. 1.23, 010 == 8, 0xA1 == 161

int, **float**

For type casting between integers and floats. E.g. (int)3.45 == 3, (float)3 == 3.000000

<, **>**, **=**, **-**, **+**, *****, **/**, **%**, **(** ,**)**

Mathematical operators. Note that '==' should be used for equality. E.g. 10 % 4 == 2 , (1 + 2) * 3 == 9

&, **|**

Logical operators. Note that '&&' and '||' should be used for 'and' and 'or', that '1' equals 'true' and '0' equals 'false'. E.g. 1+0 && 1

If any other character is used there will be an error in the RXML parser. Therefore entities should be used, not the variable name, in a similar way as *Match*.

```
<set variable='var.foo' value='2' />

<if expr='1+&var.foo;*3==7'>
   var.foo = 7
</if>
<else>
   var.foo = 9
</else>
```

Results in:

```
var.foo = 7
```

An example of expr showing that the regular rules of mathematics applies here. The expression evaluates correctly as

```
1+2*3 == 1+6 == 7
```

and not as

```
1+2*3 == 3*3 == 9
```

Note that the '==' operator must be used for 'equals', unlike the other Eval plugins. Also note that whitespaces around operators are not mandatory.

Clientvar extends the *Supports* plugin, see State below. It is used for tests of the client requesting the web page, e.g. a browser or a WAP phone. The following variables (var) are currently testable:

- height - The presentation area height in pixels (WAP clients)
- javascript - The highest version of javascript supported
- robot - The name of the web robot
- width - The presentation area width in pixels (WAP clients)

is can be any valid operator.

```
<if supports='javascript'>
<if clientvar='javascript < 1.2'>
    Your browser supports older versions of javascript
</if>
<else>
    Your browser supports javascript
</else>
</if>
<else>
  Your browser doesn't support javascript at all
</else>
```

The variable can be used in expressions as shown above. *Clientvar* can test the exact JavaScript version supported, unlike *Supports*, that only checks if JavaScript is supported or not.

2.3.3 Match

The *Match* category contains plugins that match contents of something, e.g. an IP package header, with arguments given to the plugin as a string or a list of strings.

Plugin	Syntax
Accept	accept='type1[,type2,...]'
Client	client='pattern'
Domain	domain='pattern1[,pattern2,...]'
Ip	ip='pattern1[,pattern2,...]'
language	language='language1[,language2,...]'
Referrer	referrer='pattern1[,pattern2,...]'

Accept checks which content types the browser accepts as specified by it's Accept-header, e.g. 'image/jpeg' or 'text/html'.

```
<p>You are using
<if client='Mozilla*'>
<if client='*compatible*msie*'>
<b>Internet Explorer</b>.
</if>
<elseif client='*compatible*opera*'>
<b>Opera</b>.
</elseif>
<else>
<b>Netscape</b>.
</else>
</if>
<else>
  another client.
</else>
</p>
```

Client compares the user agent string with the given pattern.

Domain and Ip plugins checks if the DNS name or IP address of the user's computer match any of the patterns specified. Note that domain names are resolved asynchronously, and that the first time the browser accesses a page containing <if domain>, the domain name might not have been resolved yet.

Language matches languages with the Accept-Language header.

Referrer checks if the Referrer header match any of the patterns.

2.3.4 State

State plugins check which of the possible conditions something is in, e.g. if a flag is set or not, if something is supported or not, if something is defined or not etc.

Plugin	Syntax
Config	config='name'
False	false=''
Pragma	pragma='string'
prestate	prestate='option1[,option2,...]'
supports	supports='feature'
True	true=''

True and *False* are plugins used in exactly the same way as `<then>` and `<else>`. Should not be confused with the `<true>` and `<false>` tags.

```
<set variable='var.foo' value='10' />

<if variable='var.foo is 10' />
<if true=''>
  var.foo is 10
</if>
<if false=''>
  var.foo is not 10
</if>
```

Results in:

```
var.foo is 10
```

Config tests if the RXML config named has been set by use of the `<aconf>` tag.

```
<if pragma='no-cache'>
  The page has been reloaded!
</if>
<else>
  Reload this page!
</else>
```

Pragma compares the HTTP header pragma with the given string.

Prestate checks if all of the specified prestate options, e.g. '(debug)', are present in the URL.

Supports tests if the client browser supports certain features such as frames, cookies, javascript, ssl among others. An example of this with a complete list of features that can be tested is found in the section Browser independency with <if supports>.

2.3.5 Utils

The *Utils* category contains additional plugins, each specialized in a certain type of test.

Plugin	Syntax
Date	date='yyyymmdd' [before,after] [inclusive]
Exists	exists='path'
Group	group='name' groupfile='path'
Time	time='hhmm" [before,after] [inclusive]
User	user='name1[,name2,...] [any]'

```
<if time='1200' before='' inclusive=''>
   ante meridiem (am)
</if>
<else>
   post meridiem (pm)
</else>
```

Date and *Time* plugins are quite similar. They check if today is the date "yyyymmdd" or if the time is "hhmm". The attributes *before*, *after* and *inclusive* may be added for wider ranges.

```
<if exists='/dir1/dir2/foo.html'>
   foo.html exists
</if>
<else>
   foo.html doesn't exist
</else>
```

Results in:

```
foo.html doesn't exist
```

Exists checks if a file path exists in a file system. If the entered path does not begin with '/', it is assumed to be a URL relative to the directory containing the page with the `<if exists>` statement.

Group checks if the current user is a member of the group according the groupfile.

A useful *Util* plugin is *User*, that tests if the user accessing a site has been authenticated as one of the users specified. The argument any can be given for accepting any authenticated user. More about this plugin in the Section User authentication with <if user>.

2.3.6 SiteBuilder

SiteBuilder plugins requires a Roxen CMS installed to work. They are adding test capabilities to web pages contained in sites administered via SiteBuilder.

Plugin	Syntax
Ppoint	ppoint='RXML protection point' none,read,write

The *Ppoint* plugin test if a user has access to a RXML protection point. The attributes none, read and write specify what permissions are required for the test to return true.

2.3.7 Summary

This Section has presented the If plugins that is divided into fourfive categories, *Eval*, *Match*, *State*,*Utils*,and *SiteBuilder*, according to their function.

Eval plugins evaluate expressions as in regular programming languages, *Match* plugins match contents with arguments given and *State* plugins check which of two possible conditions is met. *Utils* plugins perform specific tests such as present date or time. SiteBuilder plugins require a Roxen CMS and add test capabilities to web pages contained in a SiteBuilder.

More details about If plugins are found in the Web Site Creator Manual. Information about SiteBuilder is found in the Administration Manual.

The next Section, A basic example of <if>, will show an example of a basic usage of the *Match* plugin.

2.4 A basic example of <if>

This section will show a simple example of how we use `<if>` in a web page.

After reading this section you will understand how to use `<if>` with the *Match* plugin.

We will create a feature on the DemoLabs Inc. site (shipped with Roxen CMS). This feature includes the possiblity to toggle between short view and long view using a button while reading a protocol in the Management Protocol Archive. The short view only displays the header of the protocol, while long view shows the full protocol.

2.4.1 Short/Long feature in protocols

Imagine that the protocols tend to be rather long and that someone only wants to check which persons were present and what issuses that were discussed during a meeting. Wouldn't it be nice only to view a 'header' of the protocol per default, containing these data. The whole protocol should only be displayed when the user explicitly requests that. To accomplish this we add some RXML and a button. The button will be used to toggle between viewing the whole protocol and only the 'header'.

Since we want every new protocol file to have this feature we will edit the Stationary Protocol file protocol.html. This is a default protocol file that is used as a foundation for creating new protocol files. At the top of this file we add the following code:

```
<!-- Page loaded first time -->
<if match='&form.request; is '>
<form method='POST'>
<input type='hidden' name='request' value='long' />
<submit-gnutton gnutton='gnutton5' align='left'
>Long view</submit-gnutton>
<br /><br />
</form>
</if>

<!-- When long mode is requested -->
<elseif match='&form.request; is long'>
<form method='POST'>
<input type='hidden' name='request' value='short' />
<submit-gnutton gnutton='gnutton5' align='left'
>Short view</submit-gnutton>
<br /><br />
</form>
</elseif>

<!-- When short mode is requested -->
<else>
<form method='POST'>
<input type='hidden' name='request' value='long' />
<submit-gnutton gnutton='gnutton5' align='left'
>Long view</submit-gnutton>
<br /><br />
</form>
</else>
```

Before the line `<h1>Opening</h1>`, which is the starting header for the full view mode we add an <if> test. Only if the user requests long view the content between `<if>` and `</if>` will be sent to the browser. Finally we add `</if>` at the last line of the file.

```
<if match='&form.request; is '>
<h1>Opening</h1>
    ...
</if>
```

This will result in the following button and short form of the protocol file when loaded the first time:

Figure 5. The text displayed in the short form protocol with button added on top.

The trick is to dynamically insert the right form depending on which view is requested. This is accomplished with an if-elseif-else statement. `<if>` checks for an empty `&form.request;`. This will

only be the case the first time the page is loaded. <elseif> catches the case when long view is requested and <else> the case when short view is requested.

```
<if match='&form.request; is '>
  ...
</if>
<elseif match='&form.request; is long'>
  ...
</elseif>
<else>
  ...
</else>
```

The inserted form contains a hidden field with its request variable set to the mode that will be requested on submit and a special XSLT defined button - <submit-gnutton> - with its displayed text set inside the container. XSLT is not in the scope of this Lesson, so we will leave that with telling this is used only to get a nice look-&-feel.

```
<form method='POST'>
<input type='hidden' name='request' value='long' />
<submit-gnutton gnutton='gnutton5' align='left'
>Long view</submit-gnutton>
<br /><br />
</form>
```

You could use an ordinary submit button as well:

```
<input type='submit' value='long' />
```

Well, there it is! Finally, let's have a look at a part of the long view version of a protocol page:

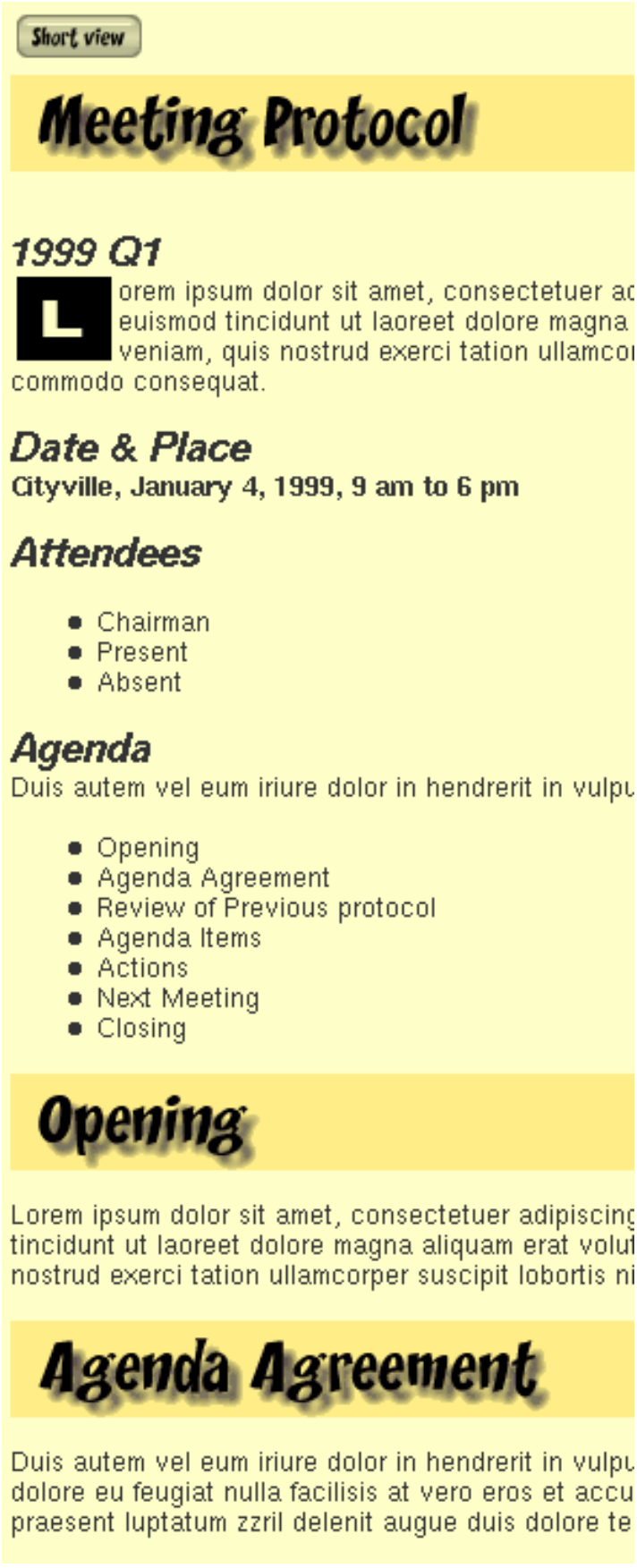

Figure 6. The button inserted on a long view protocol page.

2.4.2 Summary

This section has shown a basic example of how to use the <if> plugin *Match*.

More details about `<if>` and If Plugins are found in the Web Site Creator Manual and/or by adding `<help for='if' />` in a web page.

The next section, Combining <if> and <define> will teach the basics of how to use `<if>` and `<define>` to dynamically display contents in a web page.

2.5 Combining <if> and <define>

This section will show how to combine `<if>` and `<define>` to dynamically show contents of a web page.

After reading this section you will have knowledge of how to combine `<if>` and `<define>` when creating a web page. You will also know how to use `<define>` to define your own RXML macro.

We will create a web page with a HTML form requesting the name and e-mail address of a user. On submit we will check if all fields are set and if the e-mail address is valid. The page will show a response depending on the test results.

2.5.1 The <define> tag

A very useful tag is the *Variable* tag `<define>`. It is used for creating your own macros such as tags, containers or If-callers. The most common use is to define an 'alias' for a portion of code that will be inserted several times on a web page. We will not discuss `<define>` in depth here. See the Web Site Creator Manual for details.

Let's create a tag that will work as a sum of some other tags.

```
<define tag='multi-set'>
<set variable='var.foo' value='one' />
<set variable='var.bar' value='two' />
<set variable='var.gazonk' value='three' />
</define>

<pre>var.foo is 'var.foo'
var.bar is 'var.bar'
var.gazonk is 'var.gazonk'
</pre>

<multi-set/>

<pre>var.foo is 'var.foo'
var.bar is 'var.bar'
var.gazonk is 'var.gazonk'
</pre>
```

Results in:

```
var.foo is ""
var.bar is ""
var.gazonk is ""
 var.foo is "one"
var.bar is "two"
var.gazonk is "three"
```

Here <multi-set> is used to insert the three <set> tags. If we were to do this set operation in multiple places the value of <define> is quite obvious. We can also add values via attributes to our defined tag.

```
<define tag='hello'>
  Hello there, &_.name;!
</define>

<hello name='John Doe'/>
```

Results in:

```
Hello there, John Doe!
```

The attribute values are catched with entities and using the '_' scope, representing the current scope. Here the value of the name attribute is represented by the entity &_.name;. If we want to set default values to attributes (used when the attribute is left out) we use the container <attrib>.

```
<define tag='hello'>
<attrib name='name'>Mr Smith</attrib>
  Hello there, _.name!
</define>

<hello /><br />
<hello name='John Doe' />
```

Results in:

```
Hello there, Mr Smith!
Hello there, John Doe!
```

An interesting feature is to define a macro for an If plugin or combinations of If plugins. You simply create an alias that can be used together with <if>.

```
<define if='js'>
<if supports='javascript' and=''
      clientvar='javascript = 1.2' />
</define>

<if js='js' />
<then>
  Your browser supports javascript 1.2
</then>
<else>
  Your browser doesn't support javascript 1.2
</else>
```

Results in:

```
Your browser supports javascript 1.2
```

Here <if js='js'> is replaced by <if supports='javascript' and='' clientvar='javascript = 1.2'> before the page is sent to the browser.

Well, that is how far we go into <define> here. In the example part below we will use the <define tag> feature to insert a HTML form with dynamic content.

2.5.2 Verifying an e-mail address

We are going to create a simple e-mail address checker only using HTML and RXML. The first time the page is loaded a form is presented to the user for input of name and e-mail address. A submit sends the input and when the page is loaded again, we will first check that both name and e-mail address were added, and if so, check if the e-mail address matches the form '*@*.*'. If any of the tests fail, an error message will be displayed together with the form for new input. Correct input will not be deleted. If all goes well, a nice welcome awaits the user.

The source code is found by following the link. (It might be a good idea to open the source code file in a different window, so that it is easily read parallel to this section. The code won't be displayed here to shorten the length of this section.)

First we define a RXML macro called *mail-form* that inserts a form. Four attributes are used: *nameval*, *mailval*, *status* and *mess*. We use `<attrib>` to set default values for *nameval* and *mailval* to the data entered in the field. This will save entered data so that the user won't have to retype it. The form will display a status message, an ordinary message and a form with two input fields and a submit button.

```
<!-- DEFINING FORM TAG -->
<define tag='mail-form'>
<attrib name='nameval'>&form.name_;</attrib>
<attrib name='mailval'>&form.mail_;</attrib>

<p><b>&_.status;</b><br />&_.mess;</p>

<form method='POST'>
<table>
<tr><td>Name:</td>
<td><input type='input' size='30' name='name_'
                value='&_.nameval;' /></td></tr>
<tr><td>E-mail:</td>
<td><input type='input' size='30' name='mail_'
                value='&_.mailval;' /></td></tr>
<tr><td><input type='submit' name='button'
                value='OK' /></td></tr>
</table>
</form>
</define>
```

Figure 7. The start form.

To display the page dynamically we use <if> and <else>. The first test checks if the page is loaded for the first time or if the user pushed a submit button to get there. Only if the user clicked the 'OK' button, &form.button; will represent 'OK'. The next test checks if both fields contain data. If so, var.ok will have the value 1. The last test checks if the e-mail address match the form '*@*.*'. This test is really not sufficient in real life, since an address like 'foo@foo@mail.gazonk' would be correct. Remember that anything goes with '*'. It is left for you to figure out a nice algorithm for a better check. Remember, practice makes the master.

```
<if match='&form.button; = OK'><!-- OK clicked -->
  ...
<if variable='var.ok = 1'><!-- Both not empty -->
<if variable='var.mail_ = *@*.*'><!-- Success -->
      ...
</if>
<else><!-- Mail not on proper format -->
      ...
</else>
</if>
<else><!-- name or e-mail empty -->
    ...
</else>
</if>
<else><!-- First time or Again clicked -->
  ...
</else>
```

The first time the page is loaded or if the user clicked the 'Again' button we use our defined macro to display the start form.

```
<else><!-- First time or Again clicked -->
<mail-form status='' mess='Please state your name and
   e-mail address.' />
</else>
```

When we are sure that the user clicked the 'OK' button, we test the data entered. We use <set> to catch the data from the two input fields and then test if any of them where empty, <if variable='var.name_ = '>. If so, we use <append> to set var.ok to 0, which will give a false result in

the 'Both not emtpy' test. The form will then reappear with a message telling the user to fill in both fields.

```
<set variable='var.ok' value='1' />
<set variable='var.name_' value='&form.name_;' />
<set variable='var.mail_' value='&form.mail_;' />

<if variable='var.name_ = '>
<append variable='var.ok' value='0' />
</if>

<if variable='var.mail_ = '>
<append variable='var.ok' value='0' />
</if>

<if variable='var.ok = 1'><!-- Both not empty -->
  ...
</if>
<else><!-- name or e-mail empty -->
<mail-form status='Error' mess='You must fill in both name and
   e-mail address.' />
</else>
```

Figure 8. If any field is left empty the form reappears with an error message.

Next we check the e-mail address. We use the Variable plugin although it looks like a typical Match test. This is done to avoid the problem with whitespaces when doing a Match. (See If plugins section, Eval part for a discussion on this special case.)

```
<if variable='var.mail_ = *@*.*'><!-- Success -->
  ...
</if>
<else><!-- Mail not on proper format -->
<mail-form status='Error' mess='E-mail not on format *@*.*'
   mailval='' />
</else>
```

If the test fails the form reappears. Note that the e-mail input filed is cleared by adding `mailval=''` to `<mail-form>`.

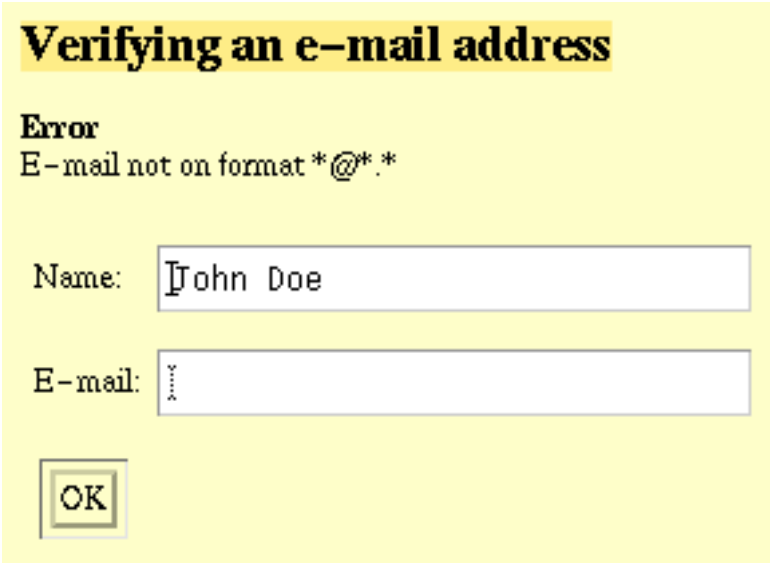

Figure 9. If any field is left empty the form reappears with an error message.

Finally, if all tests pass, we display a nice welcome message and add a button that will take the user back to the start form again.

```
<if variable='var.mail_ = *@*.*'><!-- Success -->
<p><b>E-mail address verified</b></p>
<p>Welcome <b>&form.name_;!</b></p>
<p>E-mail address '&form.mail_;' is OK.</p>
<form method='POST'>
<input type='submit' name='button' value='Again' />
</form>
</if>
```

Figure 10. If all goes well we welcome the user and display the input.

2.5.3 Summary

This section has taught you how to combine `<if>` and `<define>` to dynamically show contents of a web page.

The `<define>` tag is used for creating your own macros such as tags, containers or If-callers. The code

```
<define if='js'>
<if supports='javascript' and=''
      clientvar='javascript = 1.2' />
</define>
```

will create the macro 'js' that can be used as an alias for the <if> tag inside the container. You simply use it like this, <if js='js'></if>.

More details about <if> and <define> is found in the Web Site Creator Manual or by adding <help for='if' /> and/or <help for='define' /> in a web page.

The next Section, User authentication with <if user> will show the usage of the <if user> plugin and how to show contents in a page based on who is reading it.

The next section, Browser independency with <if supports> will teach how to use the <if supports> plugin to create dynamic pages based on what techniques the client browser supports.

2.6 User authentication with <if user>

This section will show how to use <if user> to dynamically view contents of a web page. Some useful user entities will also be discussed as well as the <user> tag.

After reading this section you will know how to create dynamic contents in a web page using <if user>. You will also know more of how to use the user entities and the <user> tag.

We will change the Marketing page of DemoLabs Inc. site (shipped with CMS Advanced) so that its content will be dynamically displayed according to which user is logged on to the site. For the examples to work, make sure that the modules 'Access Control' and 'AC: User database' are added to your own Site Builder. (For a description on how to add modules, see the Administrator manual. or contact your System Administrator.)

2.6.1 User entities and <user>

Before the example we will discuss user entities and <user> briefly. Entities are variables that are accessible everywhere if the module defining them is loaded. An entity always belongs to a scope, i.e. a group of entities providing information about something specific. To get a a listing of all present scopes you use <insert scopes='' />.

```
<insert scopes='' />
```

Results in:

```
client, cookie, form, header, modvar, page, request-header, roxen, user and
var
```

One of the scopes listed should be the user scope. This scope contains information specific to a user. A list of the entities belonging to a scope and their current values can be made available with <insert variables='full' scope='the scope'/>.

```
<pre>
<insert variables='full' scope='user'/>
</pre>
```

Results in:

```
default-workarea=0
default_workarea=0
```

```
editcols=80
editor=0
editrows=20
fullname="John Doe"
history_entries=5
localeditorfilter="none"
menu=0
nl="\n"
nojavascript=0
theme="trad"
type="user"
userid=4711
username="john"
```

Do you see your name? Most of the entities are rather SiteBuilder specific, but userid, username and fullname could be useful in a web page. At the starting page of DemoLabs, &user.fullname; is used to create a dynamic welcome message. The &user.username; entity is very useful combined with <if>, as you will see in the next part of this Section.

Before that, let's discuss the `<user>` tag.

```
<pre>
<user name='foo' />
<user name='foo' nolink='' />
<user name='foo' nohomepage='' />
<user name='foo' realname='' />
</pre>
```

Results in:

```
Foo foo@roxen.com
Foo foo@roxen.com
foo@roxen.com
Foo
```

This Information tag prints information about the specified user. By default, the full name of the user and her e-mail address will be printed, with a mailto link and link to the home page of that user. Some attributes may be added, like nolink for no links and realname for viewing only the real name of the user. For this tag to work, an authentication module is required in your Roxen CMS.

More about scopes, entities and <user> is found in the Web Site Creator Manual.

2.6.2 Dynamic Marketing page

Imagine that the Marketing page of DemoLabs Inc. site is a strategic document with information that shouldn't be read just by anyone. Some information is for customers, some is for the salesmen and some is just for the manager, Mr Smith, himself. Also imagine that we have three individuals viewing the page: Mr Smith, a salesman named John Doe and a customer called Mr Foo. There will also be an anonymous user representing the Everyone user group trying to accessing the page. We will show how to send different portions of the page to these different users. Note that in the browser output examples below only headlines will be displayed, the text is taken away for smaller snapshots.

Before we look at the code we have to get your own DemoLabs Site configured for you to be able to test the example. We have to make sure that the users mentioned above really exist (which they probably don't.) If you don't know how to perform the issues below, ask your administrator to do it for you or be satisfied with the screenshots below.

Log on to your Roxen CMS with write access to the Access Control. Go to the Access Control->Identities tab. Use the buttons to add new groups and users and set the memberships as shown in this image:

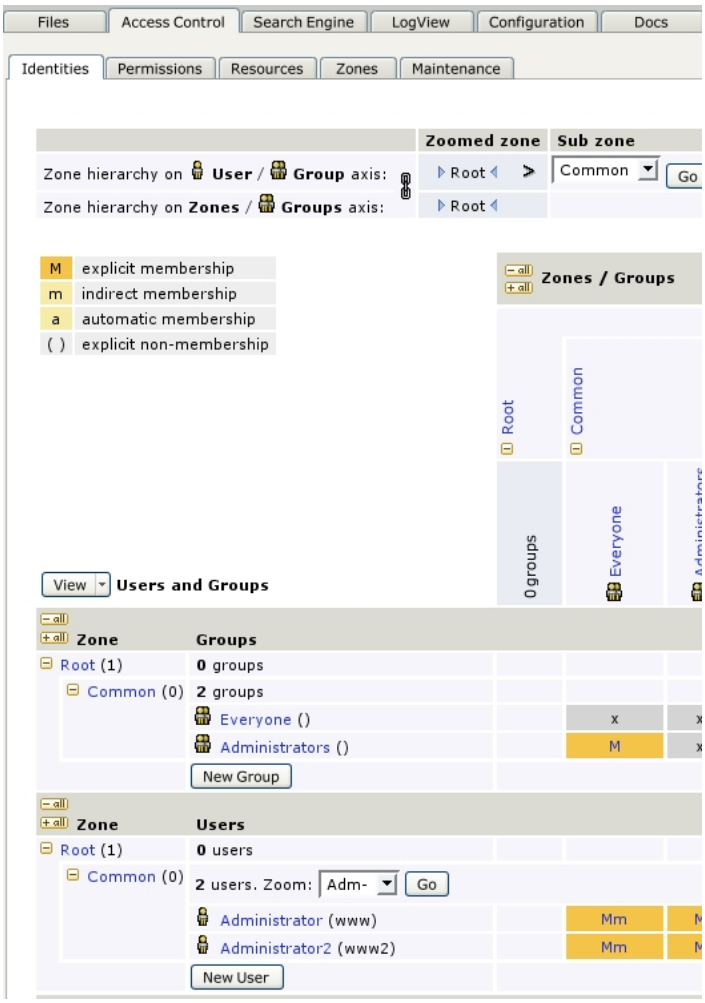

Figure 11. Users and Memberships required.

Click on the Files tab, open the Marketing directory and click Add protection point. Go back to Access Control->Permissons and set permissions according to the image below.

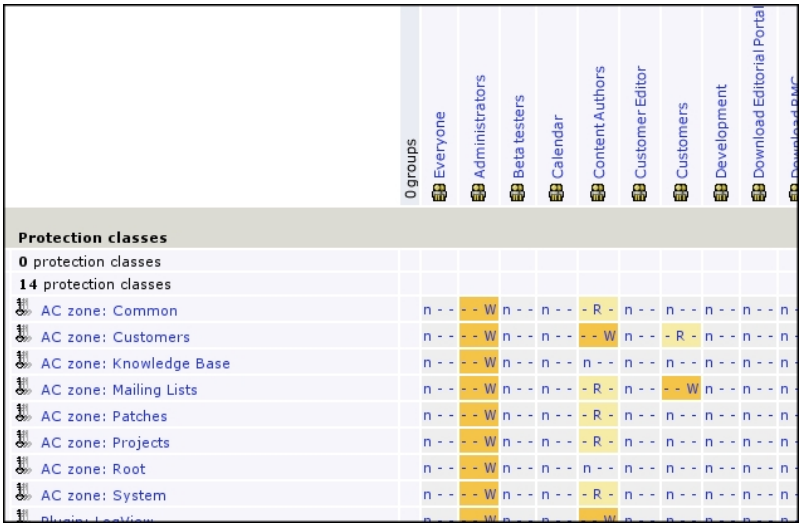

Figure 12. Permission matrix

Well, that should do the trick. Validate if you can log on as these by clicking the User button. Also note that you must committ any file for other users than the one editing it to notice the effects.

OK, on to the code. Below follows the edited code of index.xml in the Marketing directory.

```
<!-- Page displayed depending on who is logged on -->
<p><b>&user.fullname; is here!</b></p>

<if user='any'>

<!-- DISPLAYED IF USER AUTHENTICATED -->
<h1>Executive summary</h1>

<!-- ONLY SMITH AUTHORIZED -->
<if user='smith'>
<h1>Industry Analysis</h1>
<h1>Market Analysis</h1>
</if>

<!-- SMITH AND DOE AUTHORIZED -->
<if user='smith,doe'>
<h1>Marketing goals</h1>
<h1>Marketing Strategy</h1>
<h1>Marketing Tactics</h1>
</if>

<!-- ONLY SMITH AUTHORIZED -->
<if user='smith'>
<h1>Conclusions</h1>
</if>

<!-- DISPLAYED FOR ALL AUTHENTICATED USERS -->
<br />
<p><b>Written by John Smith III Jr, January 1, 2000.</b></p>

</if>

<!-- DISPLAYED FOR NON AUTHENTICATED USERS -->
<else>
<br />
```

```
<p><b>You don't have access to this document.<br/>
        Please log on first!</b></p>
</else>
```

Figure 13. This is what Mr Smith sees when he accesses the page.

The `<if user>` plugin checks if the user has been authenticated as one of the specified users. If any is given as argument, any authenticated user will do. If the user isn't authenticated, as only beeing a member of the Everyone group, the test will fail.

At the top a line is included

```
<p><b>&user.fullname; is here!</b></p>
```

telling us who is viewing the page by using the `&user.fullname;` entity.

```
<i>&user.fullname;</i> and
<i><user name='&user.username;' realname='' /></i>
shows the same thing.
```

Results in:

```
John Doe and John Doe shows the same thing.
```

The `<user>` tag with the *realname* attribute works as well, but it looks a bit more complex, doesn't it.

We encase the contents with an if-else statement to protect it from non-authenticated users. The `<if>` tag makes sure no unauthorized user sees the content. The `<else>` tag displays a message for such users. The any attribute allows all authenticated users to display the contents.

```
<if user='any'>
  ...
<else>
<br />
<p><b>You don't have access to this document.<br/>
        Please log on first!</b></p>
</else>
```

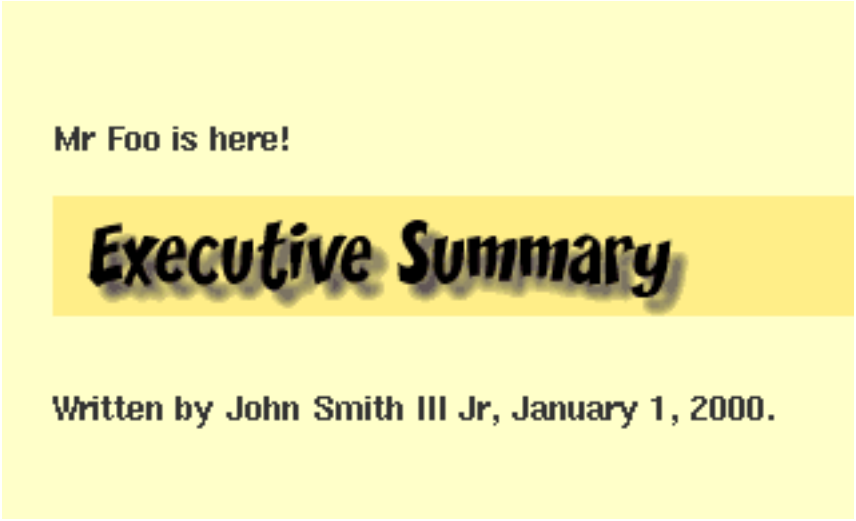

Figure 14. If a customer like Mr Foo looks at the page, only Executive Summary will be displayed...

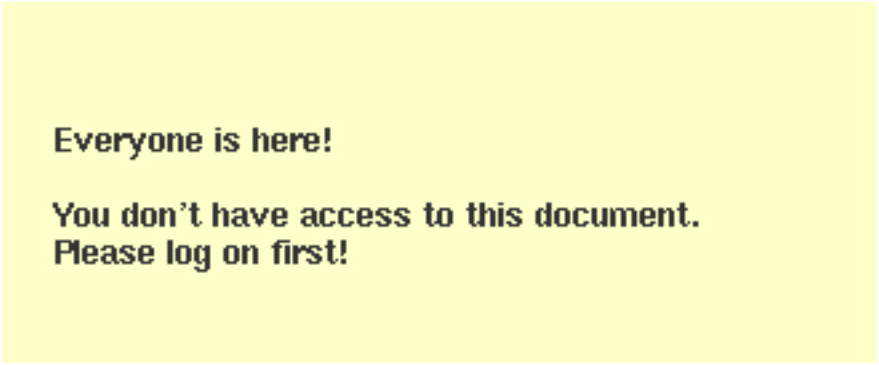

Figure 15. ...and if an unauthenticated user loads the page this is the result.

Inside this outer `<if>` we add a few others to specify authorization even more. First we restrict the contents only available to Mr Smith.

```
<if user='smith'>
<h1>Industry Analysis</h1>
<h1>Market Analysis</h1>
</if>

...

<if user='smith'>
<h1>Conclusions</h1>
</if>
```

The test could also be done with an <if match>.

```
<if match='&user.username; is smith'>
```

However, in a test for multiple users the `<if user>` tag is superior. We add a line for also allowing the salesman Mr Doe to read some parts. We use a comma separated list for the users we want to grant access to (smith and doe). A warning! Don't add any spaces like `<if user="smith, doe">` to the list. If so, only the first user (smith) will be accepted.

```
<if user='smith,doe'>
<h1>Marketing goals</h1>
<h1>Marketing Strategy</h1>
<h1>Marketing Tactics</h1>
</if>
```

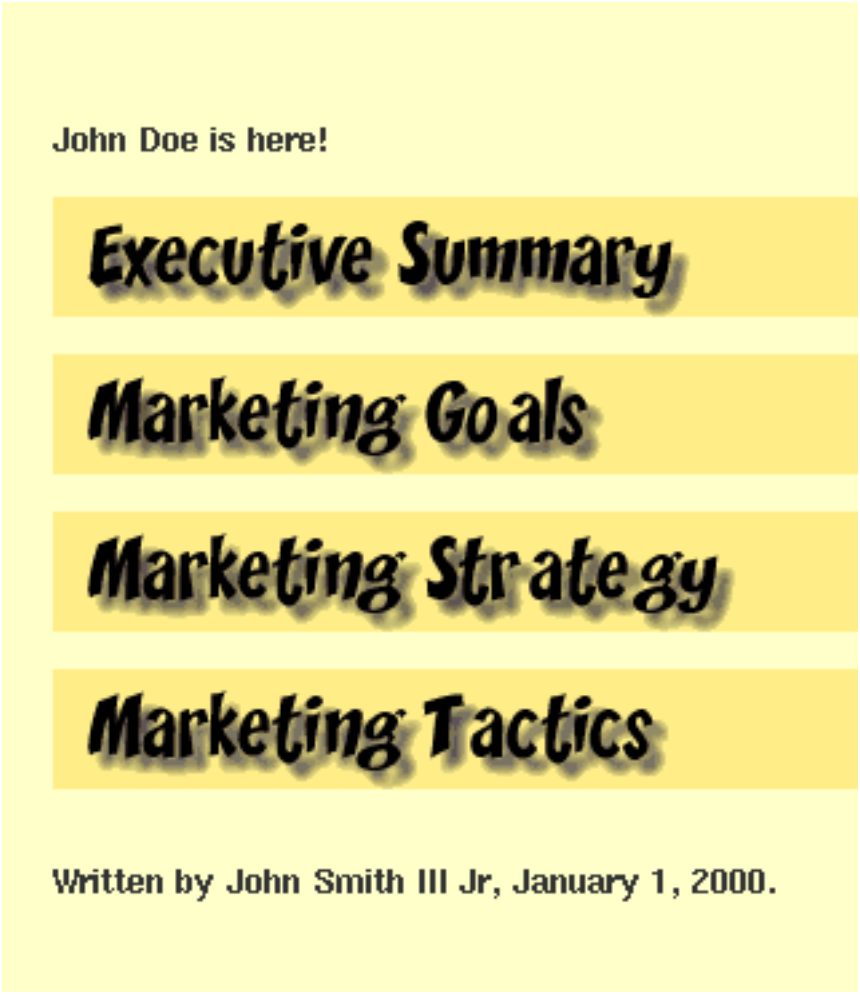

Figure 16. These are the headlines seen by the salesman John Doe.

Well, there it is. Also note that `<if group>` is available for testing if a user is authenticated as a member of the specified group, useful with a long list of users. Please see the *Web Site Creator Manual* for more information.

2.6.3 Summary

This section has taught you how to view web page contents dynamically depending on which user is requesting it. The `<if user>` tag checks if the user is authenticated as one of the users specified. With the any attribute added, it only checks if the user is authenticated or not. The entity `&user.fullname;` displays a users realname when put on a page. The tag `<user name='&user.username;' realname=''>` does the same.

For details about how to administrate a Site Builder, see the Administration Manual. Also see the Web Site Creator Manual for further information on `<user>`, `<if user>` and entities or place a `<help for='user' />` or `<help for='if' />` tag in a web page.

The next section, Browser independency with `<if supports>` will teach how to use the <if supports> plugin to create dynamic pages based on what techniques the client browser supports.

2.7 Browser independency with <if supports>

This section will deal with the `<if supports>` plugin and the page and *client* scopes.

After reading this section you will know about the many features that might be checked for with `<if supports>` and the properties of the page and client scopes and their entities.

We will create a feature on the R&D page of DemoLabs Inc. site (shipped with CMS Advanced) that dynamically uses JavaScript or HTML forms depending on the client browsers support for the JavaScript technique.

2.7.1 The <if supports> plugin features

Have you ever met somebody that have produced a good looking web page for one browser and later found out that another browser smashes it up totally? With RXML you can easily check which features the browser requesting a page supports and send the content corresponding to that. You simply use the <if supports> plugin. Below follows a list of the different features this If plugin is able to check.

Attributes

tests if the HTML attribute is supported inside tags. Possible features:

```
align
backgrounds
fontcolor
imagealign
mailto
tablecolor
tableimages
```

Client type

checks if the browser is of a certain client type, e.g. msie checks if the browser is an Internet Explorer. Possible features:

```
html
msie
phone
robot
unkvown
```

Graphics

is graphic related tests, i.e. if the browser supports certain image formats. Possible features:

```
gifinline
jpeginline
pjpeginline
pnginline
wbmp0
```

Tags

checks if the browser supports these HTML tags at all and/or in a proper manner. Note that divisions and div have the same function. Possible features:

```
bigsmall
center
divisions/div
font
forms
frames
images
layer
```

```
math
noscript
supsub
tables
```

Techniques

tests if the browser supports a certain technique. Possible features:

```
activex
autogunzip
cookies
java
javascript
js_image_object
js_inner_html
pull
push
ssl
stylesheets
vrml
wml1.0
wml1.1
```

The syntax for Supports is

```
<if supports='feature'>
```

where *feature* is replaced by one of the keywords above. The `javascript` attribute is used to test if the browser supports JavaScript or not. It will be used in the example part below.

```
<if supports='javascript'>
<if clientvar='javascript < 1.2'>
    Your browser supports javascript versions less than 1.2
</if>
<else>
    Your browser supports javascript version 1.2 or higher
</else>
</if>
<else>
  Your browser doesn't support javascript at all
</else>
```

If you want a more exact test for which JavaScript version is supported, you can use `<if clientvar="javascript is version">`, where is can be replaced by any valid operator. This is not used in the example part below.

2.7.2 Page and client scopes

For displaying and/or getting information about the client browser the client scope is suitable. The following list is an extract of the many different entities available:

- accept-language - The preferred language of the client, e.g. 'en'.
- accept-languages - The preferred language and a list of other languages also accepted.
- fullname - The full user agent string, e.g. 'Mozilla/4.7 [en] (X11; I; SunOS 5.7 i86pc)'
- ip - The ip address the client is located at.
- javascript - The javascript version supported by the client.
- name - The name of the client, e.g. 'Mozilla/4.7'
- referrer - The URL of the page on which the user followed a link to this page.

If information about a page is wanted, there is a convenient scope called page for such tasks. Some useful entities are:

- bgcolor/fgcolor - the background/foreground color of the page
- description - as specified in meta data.
- filename
- filesize - in bytes.
- keywords - as specified in meta data.
- title - as specified in meta data.
- type - the content type of the file, e.g. 'text/html'.
- url - the URL to this file from the web server's point of view.

For the complete list of entities of these scopes, insert

```
<insert variables='full' scope='client' />
  or
<insert variables='full' scope='page' />
```

in a web page. Let's look at some examples:

```
<if supports='msie'>
  You are using Internet Explorer
</if>
<elseif match='&client.name; is Mozilla/4.*'>
  You are using Netscape 4.*
</elseif>
<else>
  You are using &client.name;
</else>
</before>
```

A simple check for which browser is requesting this page.

```
<if supports='javascript'>
  JavaScript version: &client.javascript;
</if>
<else>
  Doesn't support JavaScript.
</else>
```

An example of the use of javascript attribute and entity. The &client.javascript; entity contains the actual version supported. Clientvar plugin can be used for narrow test for JavaScript version supported. See the If plugins Section, Eval part, for details and an example.

2.7.3 Browser JavaScript support optimizing

To demonstrate how to use the <if supports> plugin, we will create the possibilty to contact the R&D team of DemoLabs Inc. by submitting a message written in a form. The contact form will be displayed in a pop-up window and the message will be sent back to the parent window on submission. This is nicely done by adding some JavaScript code. However, some browsers don't support JavaScript or has it turned off. Therefore we will test if the browser supports our script, and if not, we will bring the user a pure HTML form instead.

The <if supports='javascript'> works like the HTML <noscript> tag. There is one major difference, though; <if supports='javascript'> doesn't catch if the browser has JavaScript turned off. On the other hand, <if supports> works on all browsers, also those where <noscript> doesn't (like Internet

Explorer 2.0 and Netscape Navigator 2.0). We will show how to combine these in a very useful manner.

The source code of the files in this example is found by following the link. (It might be a good idea to open the source code file in a different window, so that it is easily read parallel to this Section. The code won't be displayed here to shorten the length of this Section.) Remember that this is a RXML tutorial. Although we use JavaScript and XSLT (Extensible StyleSheet Language Transformer) code, we won't explain that in detail here. If you aren't familiar with those techniques, don't dig into that code. We will add enough information for you to understand this Section anyway.

OK, let's get down to work.

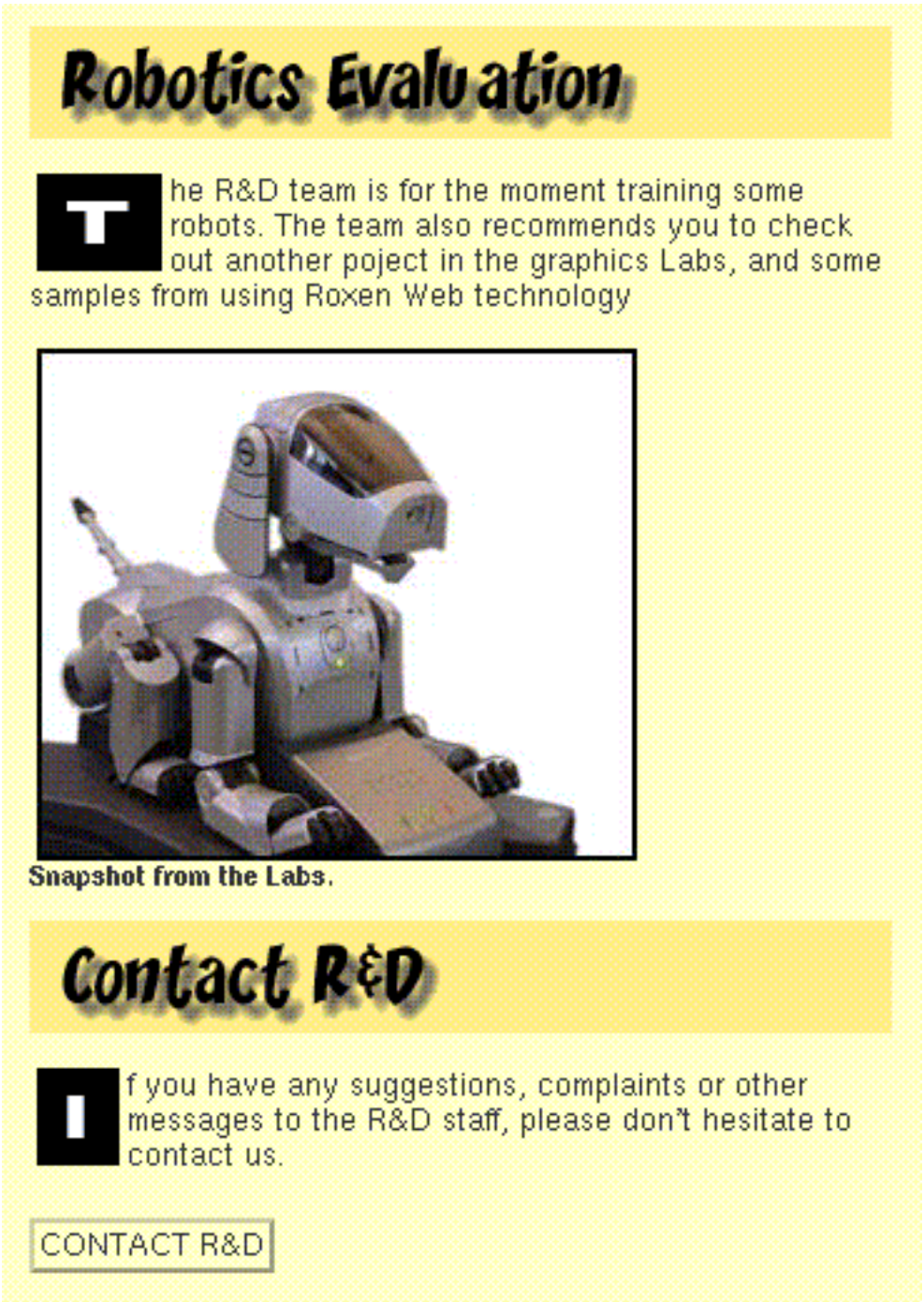

Figure 17. This is the JavaScript version of index.xml.

The snapshot above shows the JavaScript version of the edited index.xml of the R&D directory. The 'Contact R&D' part with a button is added. Let us look how this is done.

```
<h1>Contact R&D</h1>
<p initial='initial'>If you have any suggestions,
 complaints or other messages to the R&D staff,
 please don't hesitate to

<if supports='javascript'>
  ... <!-- supported -->
<noscript>
    ... <!-- turned off -->
</noscript>
</if>
<else>
  ... <!-- not supported -->
</else>
```

First we add the headline, some text and an if-else statement that will check if JavaScript is supported and on or not. The supported section will contain the JavaScript version code, the turned off will display a message saying that this page uses JavaScript, please turn this feature on or use the HTML version. Not supported section will contain the HTML version code.

Figure 18. The <noscript> version contents added instead.

```
<noscript>
<p><b>This page uses JavaScript,
   so you should enable JavaScript in your browser options
   and reload this page for the button above to work.
   Else, <a href='message.xml'>click here</a>
   to contact R&D staff.</b></p>
</noscript>
```

The <noscript> version, sent when we know that the browser supports JavaScript but has this feature turned off, gives the user an opportunity to choose version. The link leads to the contact form in pure HTML. The <else> code, sent when the browser really doesn't support JavaScript, looks like this:

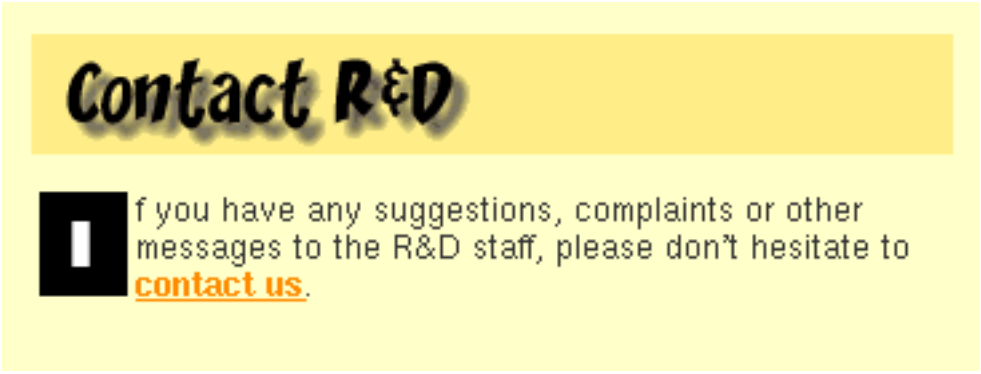

Figure 19. For browsers not supporting JavaScript this is added.

```
<else>
<a href='message.xml'><b>contact us</b></a>.
</else>
```

We simply insert a link leading to the message form in HTML version. It is a basic form with no tests of input or other funny stuff. (We don't comment the form here. Have a look in the source code if you are curious. The file is named message.xml.)

Figur 1. The message form in HTML version.

OK, this was (hopefully) the exceptions when users view the page. Let's consider the JavaScript version again.

```
<if supports='javascript'>
  contact us.
<form>
<input type='button' value='CONTACT R&D'
     onClick='openMessWin('messForm.xml')' />
</form>

<form name='hiddenForm' action='disp_mess.xml'>
<input type='hidden' name='_name' />
<input type='hidden' name='mail' />
<input type='hidden' name='mess' />
</form>

<noscript>
    ...
```

```
</noscript>
</if>
```

We insert a button that opens a new window using onClick='openMessWin('messForm.xml')'. The contents of that window is coded in messForm.xml. If you have a look at the source code you see that some features are added compared to the HTML version message form - additional buttons and some JavaScript logic checking the input. (The hidden form is added for some JavaScript magic that we won't explain here. The nice look&feel is possible by the popup.xsl template file, but that is another story.)

Figure 20. This is the pop-up window that appears when the user clicks the 'CONTACT R&D' button.

The user adds name, mail address and a message, clicks 'SEND' and the pop-up window disappears and data is submitted and handled in some way by the server. In this example we simply display it in the browser. The message will be displayed the same way if the HTML form is used instead.

Figure 21. If_tags, Browser support optimizing, img 6.

Well, that's it. This was just a simple example of the powers of <if supports>. As shown in the former parts, there are many features that may be checked for and dynamically handled.

2.7.4 Summary

This section has taught you how to check for which facilties the browser requesting the web page supports and how to adjust the sent information according to that. The page and client scopes where also discussed. The syntax for supports is

```
<if supports='feature'>
```

For displaying and/or getting information about the client browser the client scope is suitable, e.g. the &client.javascript; entity holds the JavaScript version supported by the client. The page scope gets information about a page, such as &page.filename; or &page.filesize;. For the complete list of entities, insert <insert variables='full' scope='client' /> or <insert variables='full' scope='page' /> in a web page.

More details about <if supports> and client and page scopes are found in the Web Developer Manual or adding <help for="if" /> in a web page.

2.8 Summary

This lesson have been treating the Roxen Macro Language (RXML) If tags, used to create dynamic web pages based on conditions. They also make it possible to create web applications in RXML without using any programming language.

The If tags correspond to the if-else control flow statements common in regular programming languages.

If tags statements are built up by six basic tags, <if>, <else>, <elseif>, <then>, <true> and <false>. The general syntax is:

```
<if plugin1='expr' [and|or plugin2='expr' ...] [not]>
  if block
</if>
[<elseif plugin='expr' ...>
  elseif block
</elseif>]
[<else>
  else block
</else>]
```

Mandatory attribute to <if> and <elseif> is an If plugin. Logical attributes - and, or and not - adds functionality. Inside the attribute expression, '=', '==', 'is', '!=', '<' and '>' operators are valid.

For proper XML, always close tags

```
<if match='&var.foo; is foo' />
  or
<if match='&var.foo; is foo'></if>
```

and give all attributes a value

```
<if true=''>.
```

The If plugins are divided into five categories, Eval, Match, State, Utils and SiteBuilder, according to their function. Eval plugins evaluate expressions as in regular programming languages, Match plugins match contents with arguments given and State plugins check which of the conditions possible is met. Utils plugins perform specific tests such as present date or time. SiteBuilder plugins require a Roxen Platform SiteBuilder and add test capabilities to web pages contained in a SiteBuilder.

```
<set variable='var.foo' value = '1' />

<if variable='var.foo = 1'>
  foo = 1
</if>
<else>
  foo is something else
</else>
```

Results in:

```
foo = 1
```

Here is an example of a simple if-else with RXML <if> and the Eval plugin Variable.

2.8.1 References

Roxen Web Site Creator Manual

http://docs.roxen.com/roxen/2.1/creator/

Roxen Macro Language (RXML)

http://docs.roxen.com/roxen/2.1/creator/rxml/

Roxen Administrator Manual

http://docs.roxen.com/roxen/2.1/administrator/

HTML 4.01 Specification by W3C

http://www.w3.org/TR/html401/

XML 1.0 Specification by W3C

http://www.w3.org/TR/REC-xml

XSL Specification by W3C

http://www.w3.org/TR/xsl/

XSLT Specification by W3C

http://www.w3.org/TR/xslt

Netscape JavaScript Reference

http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm

If you have any questions, suggestions, comments or complaints about this lesson, please send an e-mail to manuals@roxen.com.